Université Nice Sophia Antipolis

Systèmes et Réseaux (S6)
/ L3 Miage
Cours Réseaux / 2019-20

UNIVERSITÉ CÔTE D'AZUR

CM 6 :
Couche Applications

D'après le cours de Bruno Martin et les slides du livre "Computer Networking: A Top Down Approach, 6th edition, Jim Kurose, Keith Ross, Addison-Wesley, March 2012"

Ramon APARICIO-PARDO
Ramon.Aparicio-Pardo@unice.fr

17/02/2020

PLAN CM 5-6

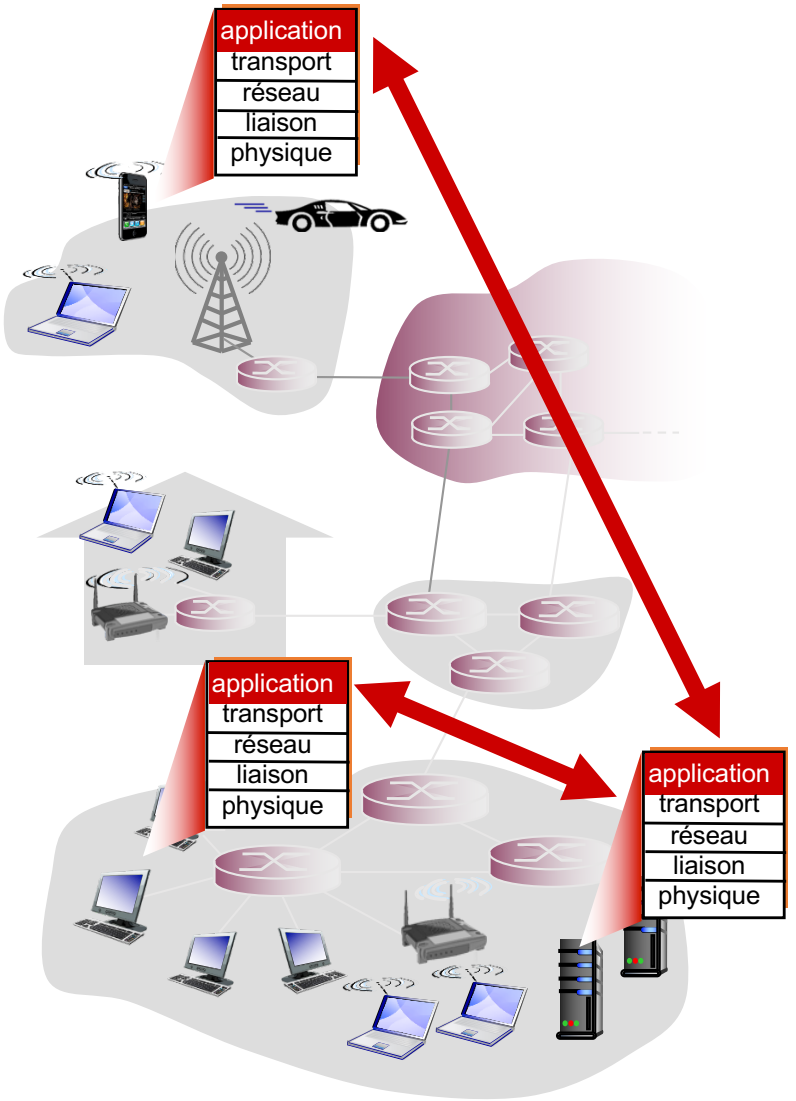
1. COUCHE D'APPLICATION
2. DNS ET REVERSE DNS
3. WEB : HTTP
4. FTP
5. COURRIER ÉLECTRONIQUE

Couche d'application

- Objectif : Concepts de base des protocoles de la couche d'application
- Des applications :
  - e-mail, web
  - text messaging, remote login, P2P file sharing
  - multi-user network games, social networking
  - streaming stored video (YouTube, Hulu, Netflix), voice over IP (e.g., Skype), real-time video conferencing
- Les protocoles les plus populaires de la couche d'application :
  - HTTP
  - FTP
  - Courriel: SMTP / POP3 / IMAP
  - DNS

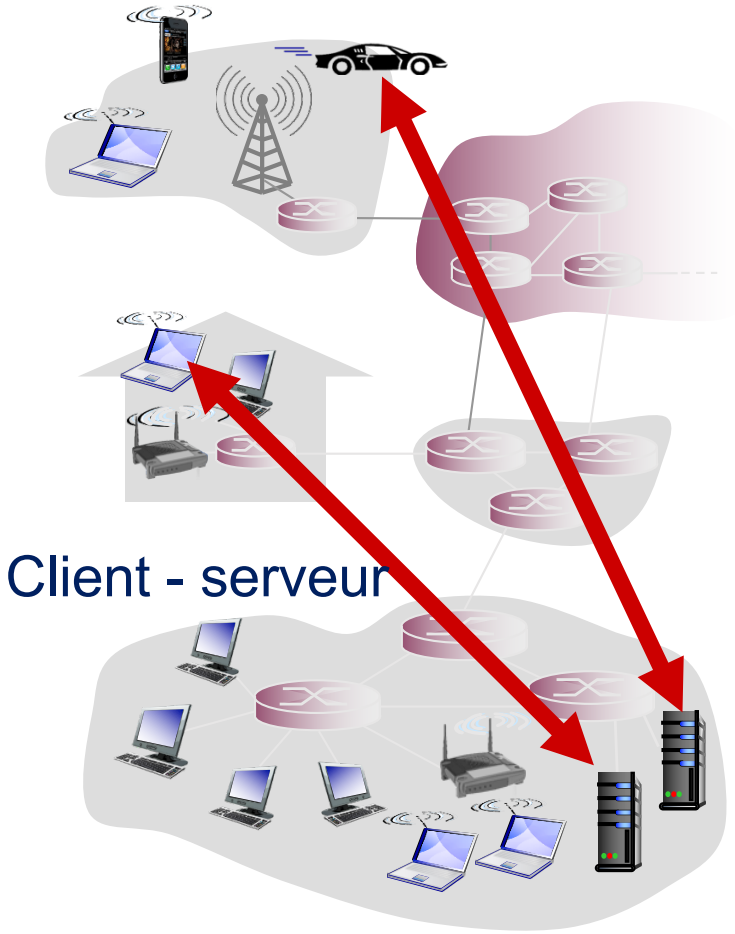
Des applications

- Une bonne application :
  - s'exécute sur différents systèmes
  - communique en réseau de bout-à-bout
  - ne s'exécute pas sur les nœuds de cœur de réseaux
  - peut être améliorée ou remplacée facilement
  - p. ex., serveur web communique avec n'importe quel navigateur
- Deux architectures basiques :
  - architecture client - serveur
  - architecture « pair-à-pair » (ou P2P, pour *peer-to-peer*)



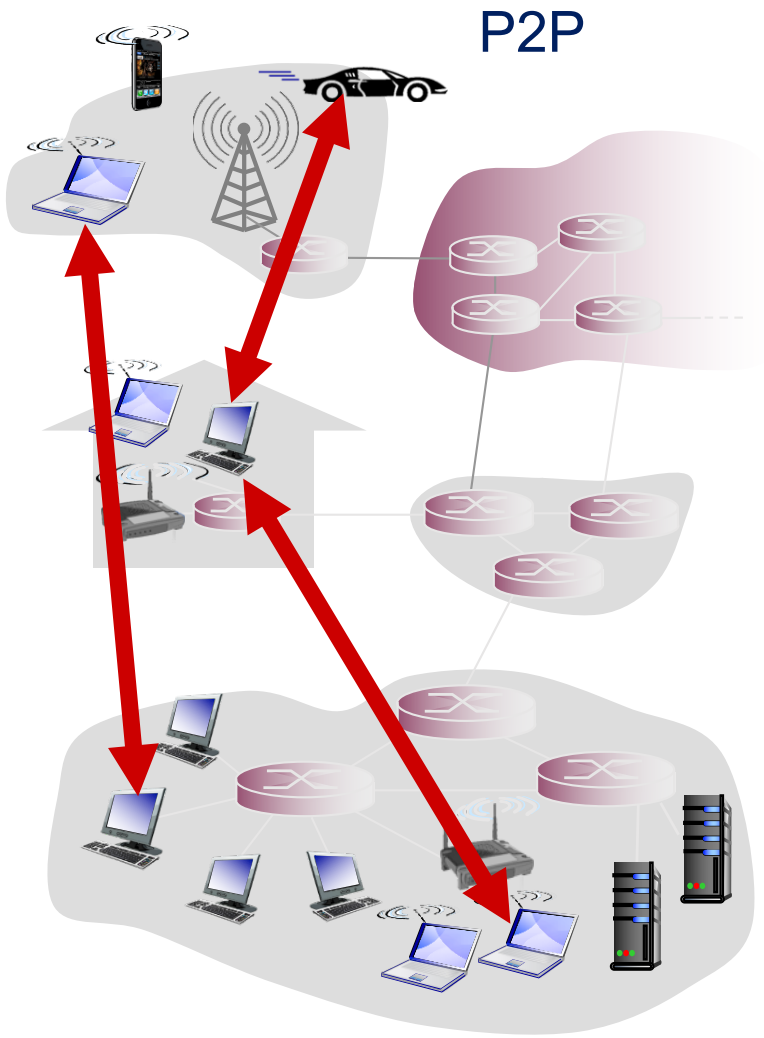
Architecture client - serveur

- Serveur :
  - toujours connecté et disponible
  - a typiquement une adresse IP permanente
  - il fournit un "service"
  - pour croître de façon durable: data-centers
- Clients :
  - communiquent avec le serveur
  - ne communiquent pas entre eux
  - connectés de manière intermittente
  - ont typiquement des adresses IP dynamiques (DHCP)
  - demandent un "service"



Architecture « pair-à-pair »

- Les pairs n'ont pas des rôles clairement définis comme dans client-serveur
  - ne sont pas toujours disponibles
  - connectés de manière intermittente
  - ont typiquement des adresses IP dynamiques
  - ils se demandent et se fournissent des "services" entre eux
  - croissance de façon durable:
    - un nouveau pair apporte de nouvelles demandes mais aussi des nouvelles capacités.
  - gestion plus complexe



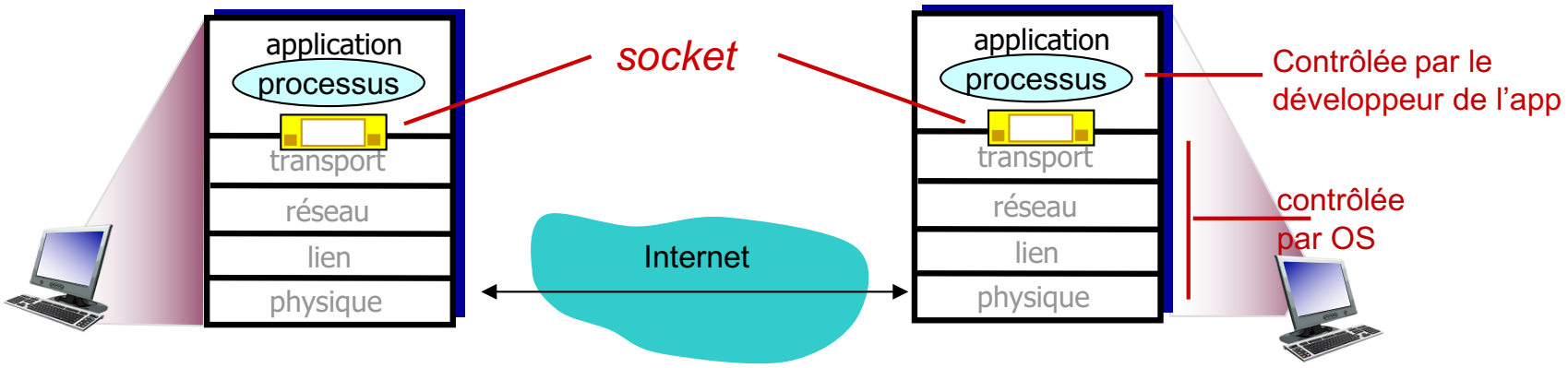
Des processus

- Un processus est un programme en cours d'exécution par un ordinateur :
  - dans la même machine, deux processus communiquent grâce à la communication interprocessus (*inter-process communication, IPC*) selon le système d'exploitation.
  - entre des machines différents, en échangeant messages

- Processus client/serveur :
  - **processus client** : processus qui démarre la communication
  - **processus serveur** : processus qui est à l'attente (à l'écoute)

Sockets

- Un processus envoie (reçoit) des messages de (vers) une sorte de identificateur: **une socket**.
  - Une socket comprend l'adresse IP de la machine plus le nº de port associé avec le processus sur la machine.
  - *Il ne suffit pas avec l'adresse IP: beaucoup de processus peuvent être exécutés sur un même hôte !!!*
  - Exemple: en se connectent au serveur web de l'université `www.unice.fr` :
    - Adresse IP: 134.59.204.9
    - Numéro de port : 80
  - Analogie avec « la poste »:
    - Le processus émetteur envoie le message vers une "adresse postale" (socket) du récepteur.
    - Le processus émetteur repose sur l'infrastructure de transport (la poste) pour livrer un message à la socket au processus récepteur



Protocoles applicatifs

- Un protocole applicative définit :
  - **les types de messages échangés** : requêtes, réponses
  - **la syntaxe des messages** : les champs dans les messages et comment ces champs sont délimités
  - **la sémantique des messages** : la signification de l'information dans les champs
  - **les règles** qui précisent quand et comment les processus envoient et répondent aux messages
- Les protocoles ouverts
  - définis dans les RFC
  - permettent l'interopérabilité
  - par exemple, HTTP, SMTP
- Les protocoles propriétaires:
  - par exemple, Skype

Les exigences applicatives sur les services de transport

- **Perte des données** :
  - Certaines applications (p. ex., transfert de fichiers, transactions Web) exigent 100% de transfert de données fiable
  - D'autres applications (p. ex., audio) peuvent tolérer une certaine perte
- **Délai (latence)** :
  - Certaines applications (p. ex., la téléphonie sur Internet, les jeux interactifs) nécessitent peu de retard pour être "efficaces
- **Débit** :
  - Certaines applications (p. ex., le multimédia) exigent un montant minimum de débit
  - D'autres applications (« apps élastiques ») font usage de tout ce qu'ils obtiennent de débit
- **Sécurité** :
  - le cryptage, l'intégrité des données, ...

application	perte de données	débit	sensibles au temps
transfert de fichiers	aucune perte	élastique	non
email	aucune perte	élastique	non
documents web	aucune perte	élastique	non
en temps réel audio/vidéo	tolérant à la perte	audio : 5Kbps-1Mbps vidéo : 10kbps-5Mbps	oui, de 100 msec
stockée audio/vidéo	tolérant à la perte	comme ci-dessus	comme ci-dessus
jeux interactifs	tolérant à la perte	quelques kbps	oui, quelques sec
messagerie texte	aucune perte	élastique	oui, quelques sec

Services des protocoles de transport

- **Service TCP** :
  - **orienté connexion** : une connexion est établie de bout en bout avant que les données soient envoyées
  - **transport fiable** : garantie que les données arriveront dans le bon ordre
  - **contrôle de flux** : l'émetteur ne submerge pas le récepteur
  - **contrôle de congestion** : l'émetteur arrête lorsque le réseau est surchargé
  - **ne fournit pas** : timing, garantie de débit minimum, sécurité

- **Service UDP** :
  - **non orienté connexion** : pas de négociation préalable à l'envoi des données
  - **transfert de données non fiable** entre l'envoi et la réception de processus
  - **ne fournit pas** : fiabilité, contrôle de flux, contrôle de congestion, timing, garantie de débit, sécurité, la configuration de connexion

application	protocole d'application	protocole de transport
email	SMTP [RFC 2821]	TCP
terminal distant	Telnet [RFC 854]	TCP
web	HTTP [RFC 2616]	TCP
transfert de fichiers	FTP [RFC 959]	TCP
streaming multimédia	HTTP (p. ex. YouTube), RTP [RFC 1889]	TCP ou UDP
téléphonie internet	SIP, RTP, propriétaire (p. ex. Skype)	TCP ou UDP

Sécurisation de TCP

- **TCP et UDP :**
  - **aucun cryptage** : mots de passe sont envoyés en clair dans la socket !!!

- **SSL** :
  - SSL est à la couche d'application
  - SSL fournit une connexion TCP cryptée, intégrité des données et authentification du point final
  - Les applications utilisent les bibliothèques SSL pour parler avec TCP
  - API des socket SSL : mots de passe sont envoyés cryptés dans la socket

Quelques ports importants

Port	TCP/UDP	Nom du service ou du protocole
7	TCP/UDP	echo
20	TCP	FTP (File Transport Protocol)
22	*TCP*	*SSH (Secure Shell)*
23	TCP	Telnet
25	TCP	SMTP (Simple Mail Transfer Protocol)
53	TCP/UDP	DNS (Domain Name System)
80	TCP	HTTP (Hypertext Transfer Protocol)
110	TCP	POP3 (Post Office Protocol)
143	TCP	IMAP (Internet Message Access Protocol)
443	*TCP*	*SSL (Secure Sockets Layer) ou "HTTPS"*
465	*TCP*	*Envoi de message pour Mail (SMTP authentifié)*
554	TCP/UDP	RTSP (Real Time Streaming Protocol)
993	*TCP*	*Messagerie IMAP SSL*
995	*TCP/UDP*	*Messagerie POP SSL*

DNS et Reverse DNS

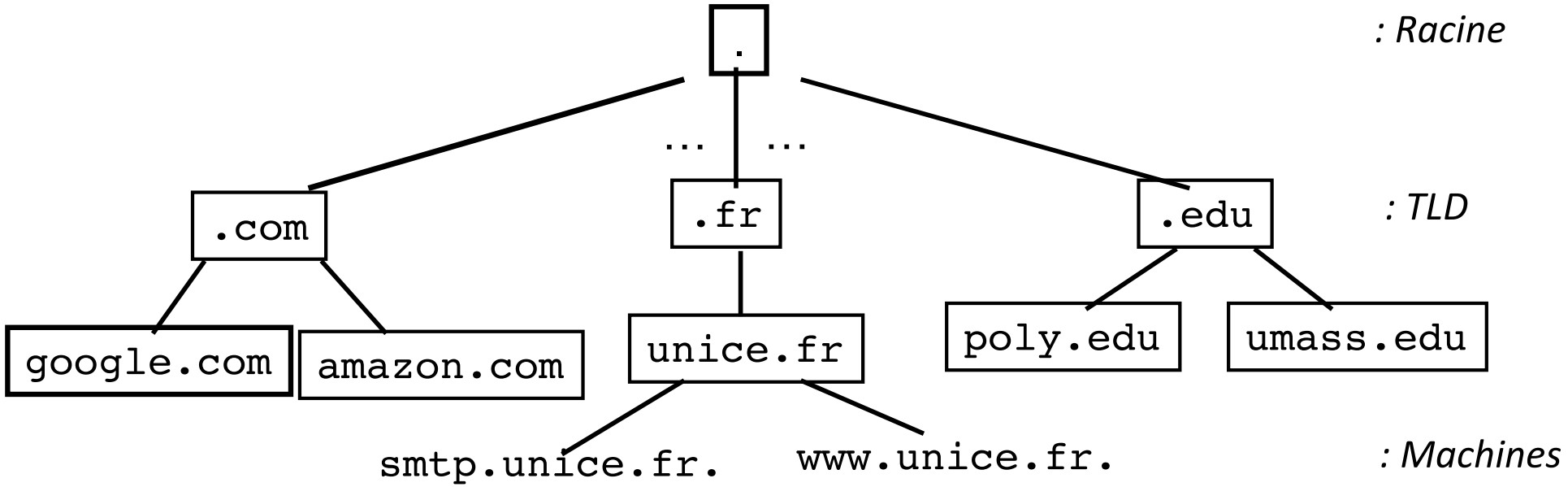
- Services DNS
  - Traduction nom d'hôte (*domain name*) ←→ Adresses IP
  - *Aliasing* de l'hôte
  - Nom canonique ←→ noms d'alias
  - *Aliasing* de serveur de messagerie
  - Répartition de la charge
  - Serveurs Web répliqués: plusieurs adresses IP correspondent à un seul nom
- Pourquoi ne pas centraliser le DNS ?
  - Point de défaillance unique
  - Volume de trafic énorme
  - Base de données centralisée distante
  - Entretien couteux
- ***Passage a l'échelle pas possible → Système décentralisé***

DNS: domain name system

- Personnes:
  - Nombreux identifiants: numéro SS, nom, passeport
- Hots Internet, routeurs :
  - Adresse IP (32 bits) - utilisée pour traiter les datagrammes
  - Nom de domaine (*domain name*), p. e. `www.unice.fr` - utilisé par les humains
- Comment associer l'adresse IP au nom, et vice versa ?
- Système de noms de domaines (*DNS: domain name system*)
  - Base de données distribuée et hiérarchique
    - Hiérarchie de nombreux serveurs de noms
  - Protocole de couche d'application :
    - Les hôtes communiquent avec les serveurs DNS pour *résoudre (resolve)* les noms (traduction d'adresse / nom)
    - Fonction principale de l'Internet, Complexité du "bord" du réseau

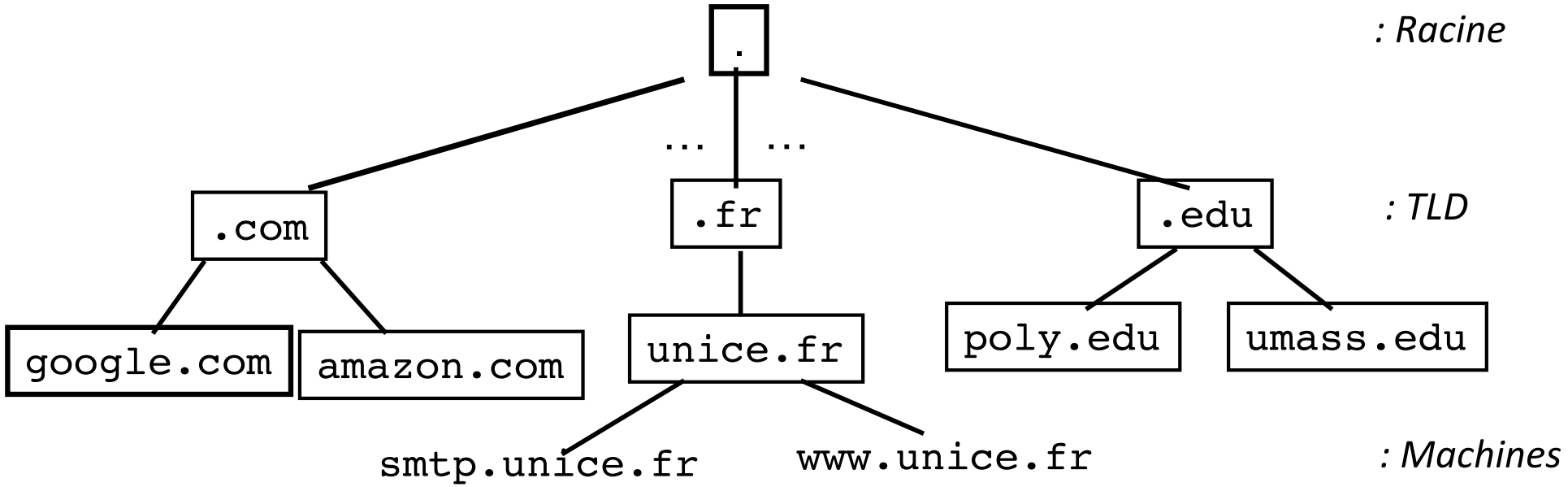
DNS: domain name system

- ***Domaine***: unité pour construction et gestion de noms
- Schéma hiérarchique par noms de ***domaines :***
  - Racine (*root*, `.`)
  - Domaines de 1° niveau (*Top Level Domain, TLD*)
    - domaines géographiques: `fr, mc, it, jp...`
    - domaines d'activité génériques: `com, org, edu, net`
  - Les autres domaines sous-jacents



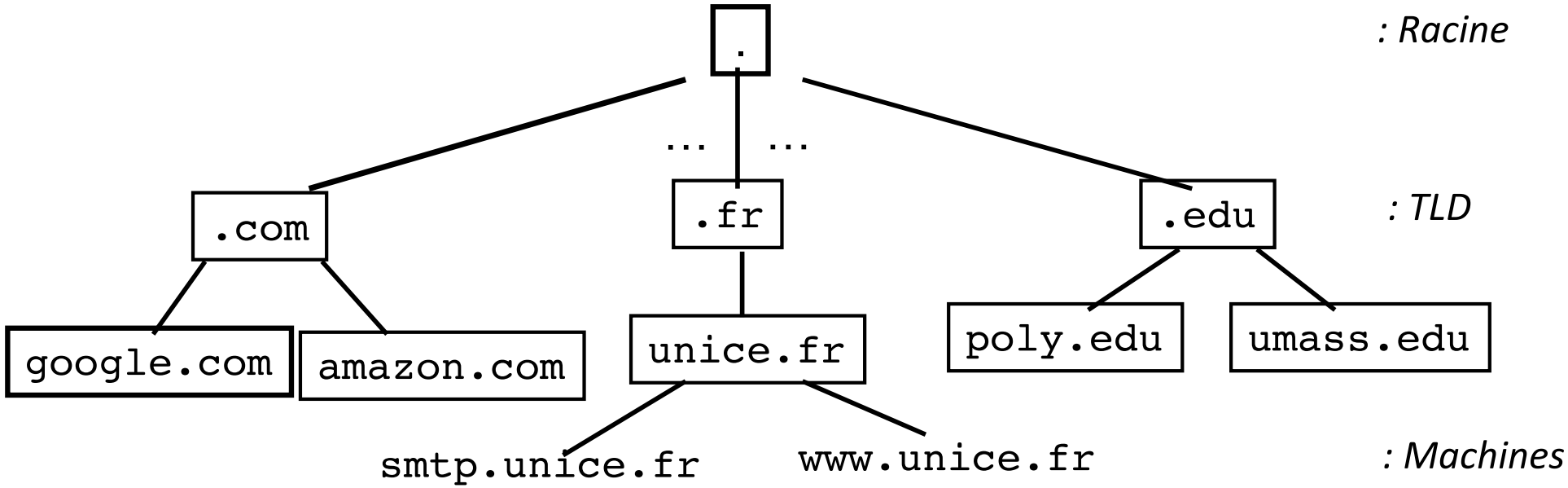
DNS: domain name system

- dans chaque domaine, noms attribués par autorité responsable
  - TLD par ISOC via groupe technique
  - domaines “publics” (`com`, `org`, `net`...) une seule autorité centrale: ICANN
  - domaines “géographiques” :
    - autorité nationale: AFNIC en France, Namebay à Monaco
  - pour les domaines inclus : autorités locales (entreprise, administration,...)



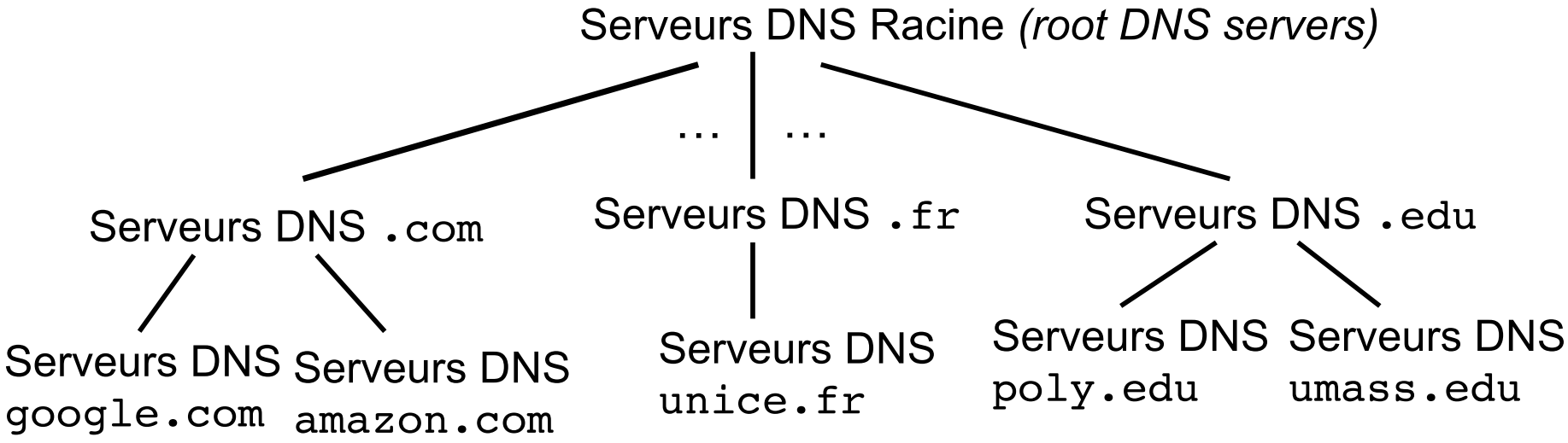
DNS: domain name system

- **Domaine** : unité de désignation (espace de noms)
- **Zone** : unité de gestion administrative (serveur de noms propre à la zone)
- Souvent, ***un domaine est une zone*** mais une zone peut regrouper plusieurs domaines administrés en commun



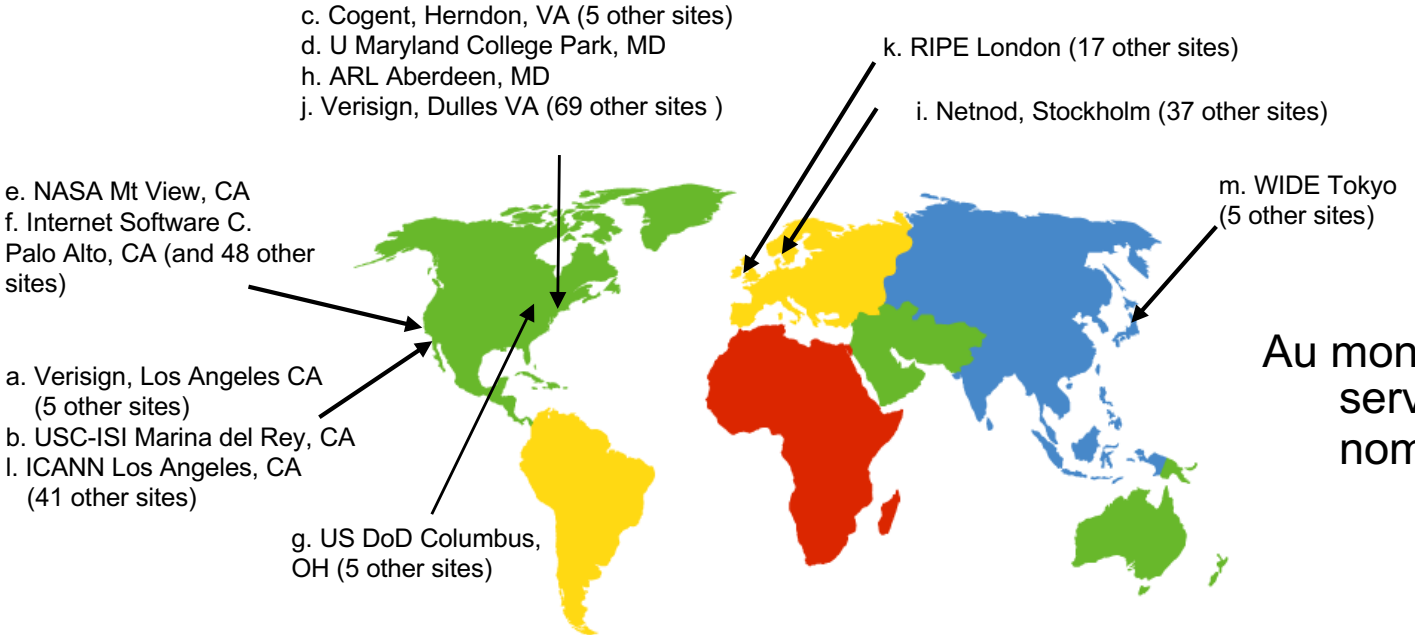
DNS: domain name system

- ***Hiérarchie de serveurs calquée sur hiérarchie de zone***
- Ex. client a besoin de l'adresse IP pour `www.unice.fr`
  1. *Client* interroge le serveur racine pour trouver un serveur DNS
  2. *Client* interroge le serveur DNS du domaine `.fr` pour obtenir le serveur DNS du domaine `unice.fr`
  3. *Client* interroge le serveur DNS du domaine `unice.f` pour obtenir l'adresse IP de la machine avec nom de domaine `www.unice.fr`



Serveurs DNS Racine

- Contacté par un serveur de noms local qui ne peut pas résoudre le nom
- Normalement, toute machine (serveur DNS) connaît leurs IPs
- Serveur de noms racine :
  - Il contacte un serveur de nom *authoritative* si le serveur racine ne connait pas la résolution nom ←→ IP
  - Obtient la résolution nom ←→ IP
  - Renvoie la la résolution nom ←→ IP au serveur de noms local



Au monde, *13* serveurs de noms racine

Serveurs DNS : TLD, authoritative, locaux

- **Serveurs de domaine TLD (*Top Level Domain*) :**
  - Responsable de com, org, net, edu, .. et tous les domaines de pays de haut niveau, par exemple: uk, fr, ca, jp
  - Ils contiennent noms de domaine du second niveau et @IP correspondantes

```
unice.fr.          86400   IN      NS      taloa.unice.fr.
unice.fr.          86400   IN      NS      dns.inria.fr.
unice.fr.          86400   IN      NS      samoa.unice.fr.
```

- **Serveur de zone (1 zone = plusieurs domaines)**
  - Ils contiennent (noms ,adresse) d'hôtes des domaines de zone et [(noms,adresse) de serveurs de sous-domaines]
  - ≥ 2 serveurs de nom par zone
  - SOA: *start of authority* : serveur de nom maître (primaire) de la zone

```
i3s.unice.fr.          86400   IN      SOA     taloa.unice.fr.
```

- ***Authoritative DNS servers :***
  - Serveurs DNS de l'organisation qui fournissant la résolution du nom pour les hôtes nommés par l'organisation
  - Peut être maintenus par l'organisation ou le fournisseur de services
  - Ils sont mis à jour en permanence : pour ça font autorité et donnent des authoritative answer

Serveurs DNS : TLD, authoritative, locaux

- **Serveur de noms DNS local**
  - N'appartient pas strictement à la hiérarchie
  - Chaque FAI (ISP résidentiel, entreprise, université) a un serveur DNS local
    - Également appelé "serveur de noms par défaut »
  - Lorsque l'hôte fait une requête DNS, la requête est envoyée à son serveur DNS local
    - Il a un cache local de résolutions IP - doms récentes de traduction (mais peut-être dépassé)
    - Il agit comme proxy, transmet la requête à la hiérarchie

Fonctionnement

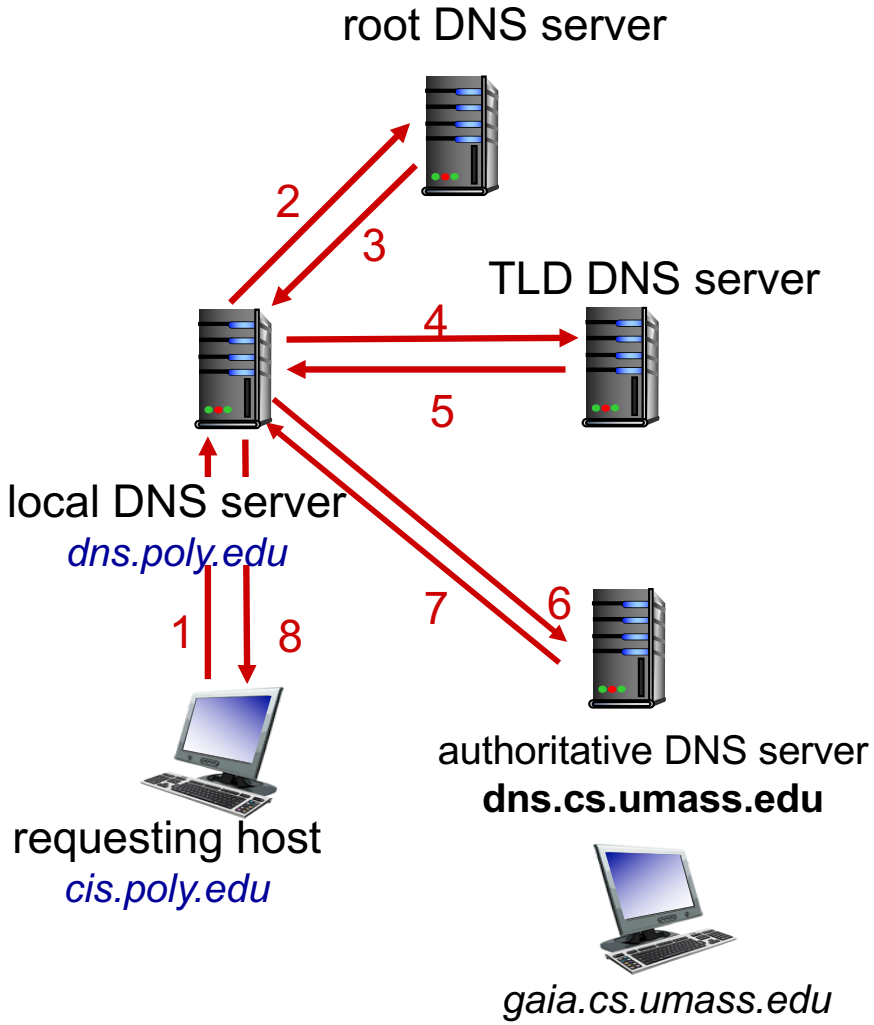
- **Accélération**: serveurs (y compris le hôte) mettent en cache (nom,@IP) récemment résolus. Mais sinon :

1. Le client envoie sa requête DNS à un serveur récursif local choisi dans le fichier `/etc/resolv.conf`. Le client essaye au besoin plusieurs serveurs.
2. Si le serveur a autorité sur la zone du domaine de la requête (*authoritative*), il répond directement avec les informations (gérées) localement
3. sinon il va rechercher le serveur cible du domaine de la requête, il relaye la requête vers ce serveur et relaye ensuite la réponse vers le client. Pour la recherche du serveur cible, on va :
   a) choisir un parent (connu localement) du domaine recherché : c'est ici soit la racine, soit le domaine que l'on gère pour une requête concernant un descendant de ce domaine local.
   b) interroger successivement les serveurs (informations NS) pour parcourir la branche entre le serveur parent et le serveur cible.
4. Cet algorithme simplifié possède deux modes de résolution : récursif et itératif.

Exemple résolution DNS

- Machine à `cis.poly.edu` veut une adresse IP pour `gaia.cs.umass.edu`

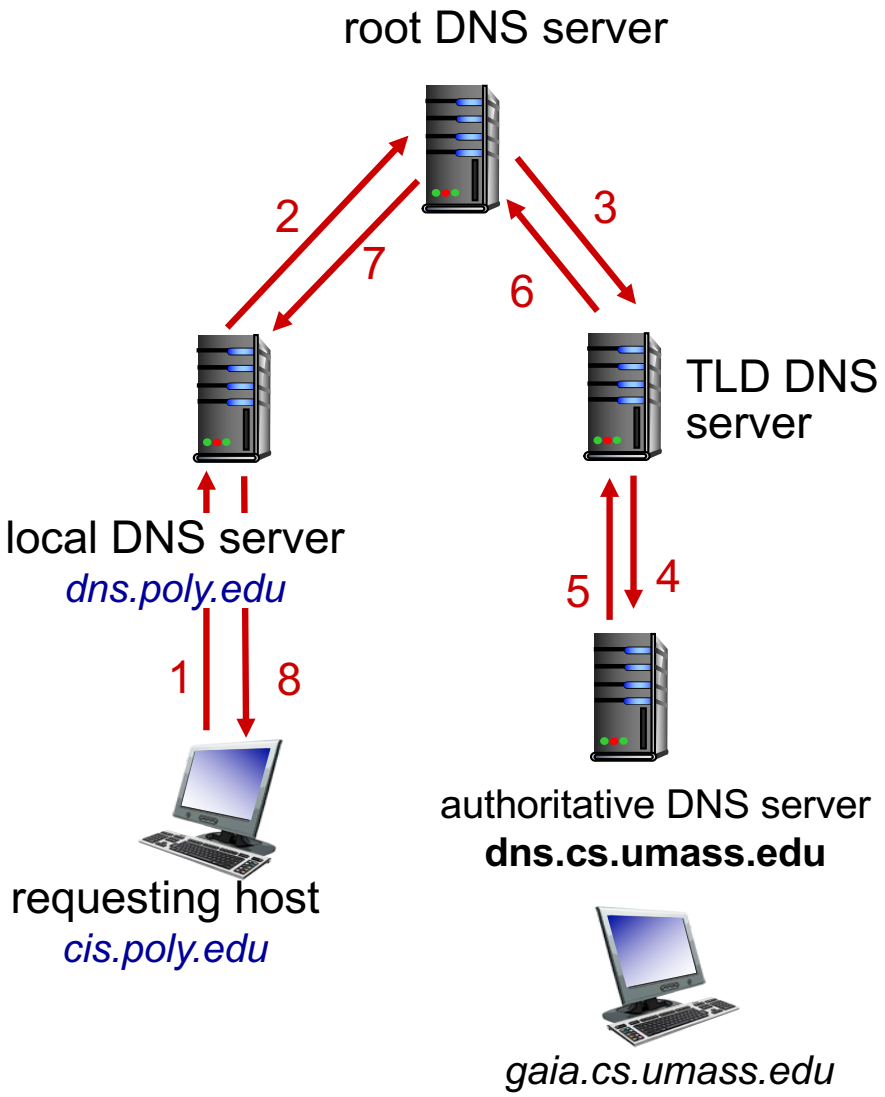
- **Mode itératif de résolution** :
  - Il consiste à demander à un serveur la meilleure information dont il dispose par rapport à la question, c'est-à-dire
    - soit la réponse
    - soit la référence à un serveur « plus proche » de la réponse : "Je ne connais pas ce nom, mais demandez à ce serveur"
  - On itère ensuite de serveur en serveur.



Exemple résolution DNS

- Machine à `cis.poly.edu` veut une adresse IP pour `gaia.cs.umass.edu`

- **Mode récursif de résolution** :
  - Il consiste à demander à un serveur de résoudre complètement la requête et de renvoyer uniquement la réponse finale.
  - Met le fardeau de la résolution des noms sur le serveur contacté
  - Charge lourde aux niveaux supérieurs de la hiérarchie
    - On passe toujours par les serveurs racine !!!



DNS: mise en cache, mise à jour des enregistrements

- Une fois qu'un serveur de noms apprend la résolution, il la met en cache
  - Expiration des entrées de cache après un certain temps (TTL)
  - Les serveurs TLD sont généralement mis en cache dans les serveurs de noms locaux
    - Ainsi, les serveurs de noms racine ne sont pas fréquemment visités

- Les entrées mises en cache peuvent être périmées (résolution *best effort*) !
  - Si l'hôte du nom change l'adresse IP, peut-être pas connu dans l'Internet jusqu'à ce que tous les TTL expirent
  - Mise à jour / notification des mécanismes proposés norme IETF [RFC 2136]

Enregistrements DNS

- DNS: base de données distribuée des enregistrements (*resource records, RR*)

Format RR: (`name, value, type, ttl`)

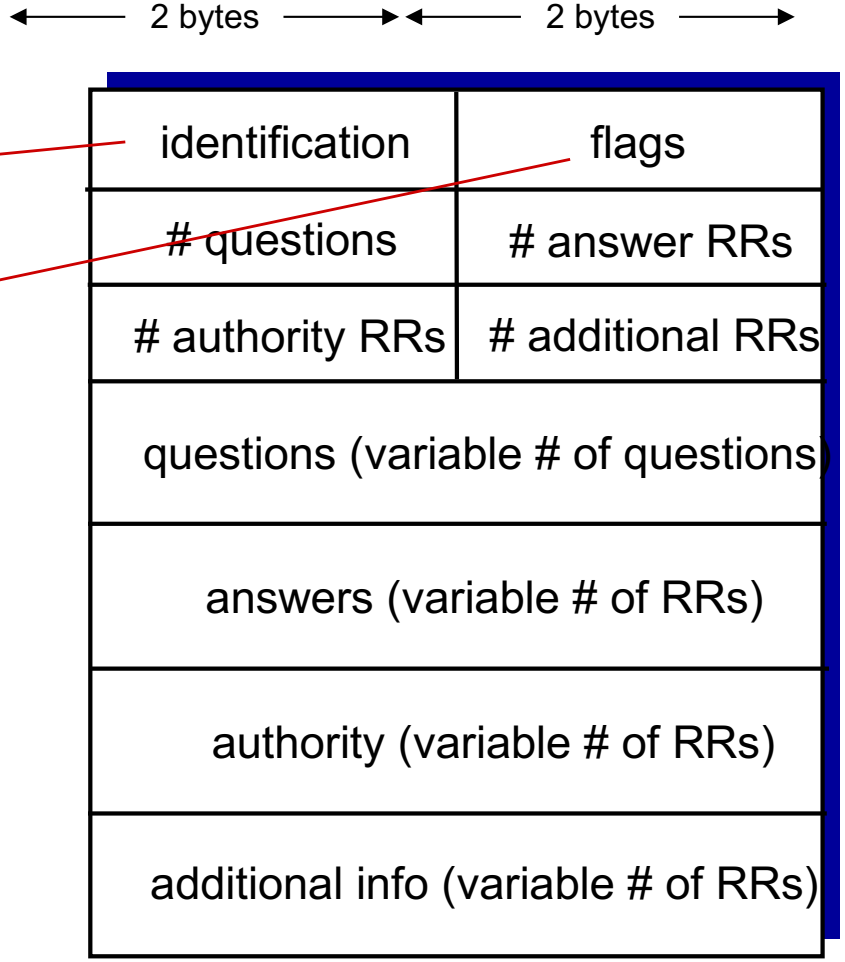
- ***type = A***
  - `name` est le nom d'hôte
  - `value` est l'adresse IP
- ***type = NS***
  - `name` est le domaine (par exemple, `unice.fr`)
  - `value` est le nom d'hôte du serveur de noms *authoritative* pour ce domaine
- **type = CNAME**
  - `name` est le nom d'alias pour le vrai nom "canonique »
  - `value` est le nom canonique
  - `name = www.unice.fr` est vraiment `value = sites.unice.fr`
- ***type = MX***
  - `value` est le nom du serveur de messagerie associé à `name`

Messages protocole DNS

- Les messages de requête et de réponse ont le même format

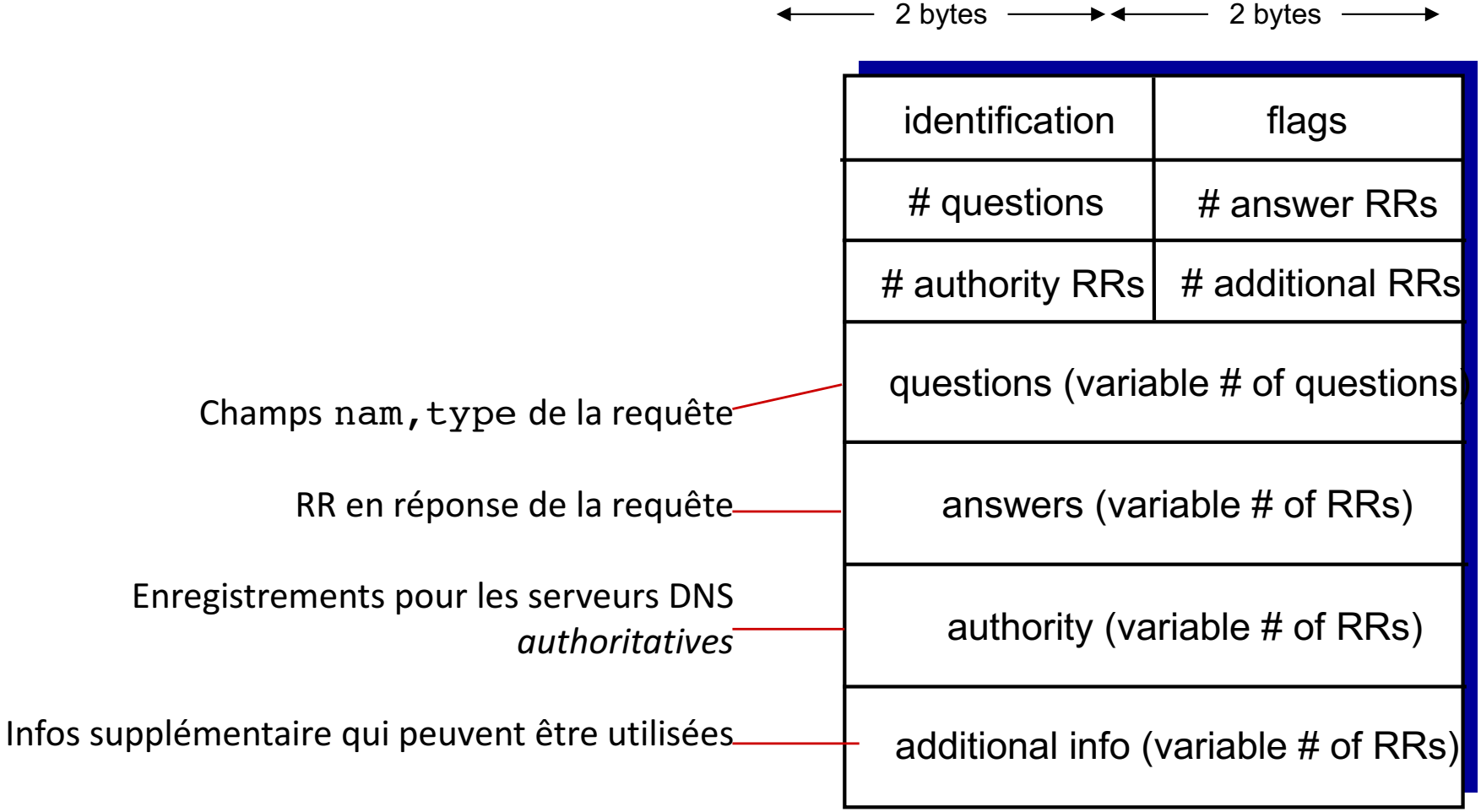
Entête message :

- Identifiant :
  - 16 bit (le même pour de requête et de réponse)
- flags:
  - requête et de réponse
  - récursivité souhaitée
  - récursivité disponible
  - réponse *authoritative*



Messages protocole DNS

- Les messages de requête et de réponse ont le même format



Insertion des enregistrements dans DNS

- Exemple: `www.unice.fr`
- inscrire le nom `unice.fr` au *registraire DNS*
  - p. ex., un registraire inscrit auprès de l'AFNIC pour le domaine `.fr`
  - Lui fournir les noms, les adresses IP de serveur de noms faisant autorité
  - registraire insère **deux** RR en serveur TLD .com :

`name = unice.fr` **`value = taloa.unice.fr`** **`type = NS`**

`name = taloa.unice.fr` **`value = 134.59.1.7`** **`type = A`**

- Dans le de serveur de noms faisant autorité (*authoritative name server*), on crée
  - un registre RR de type A pour `www.unice.fr`
  - un registre RR de type MX pour `unice.fr`

Attaquer DNS

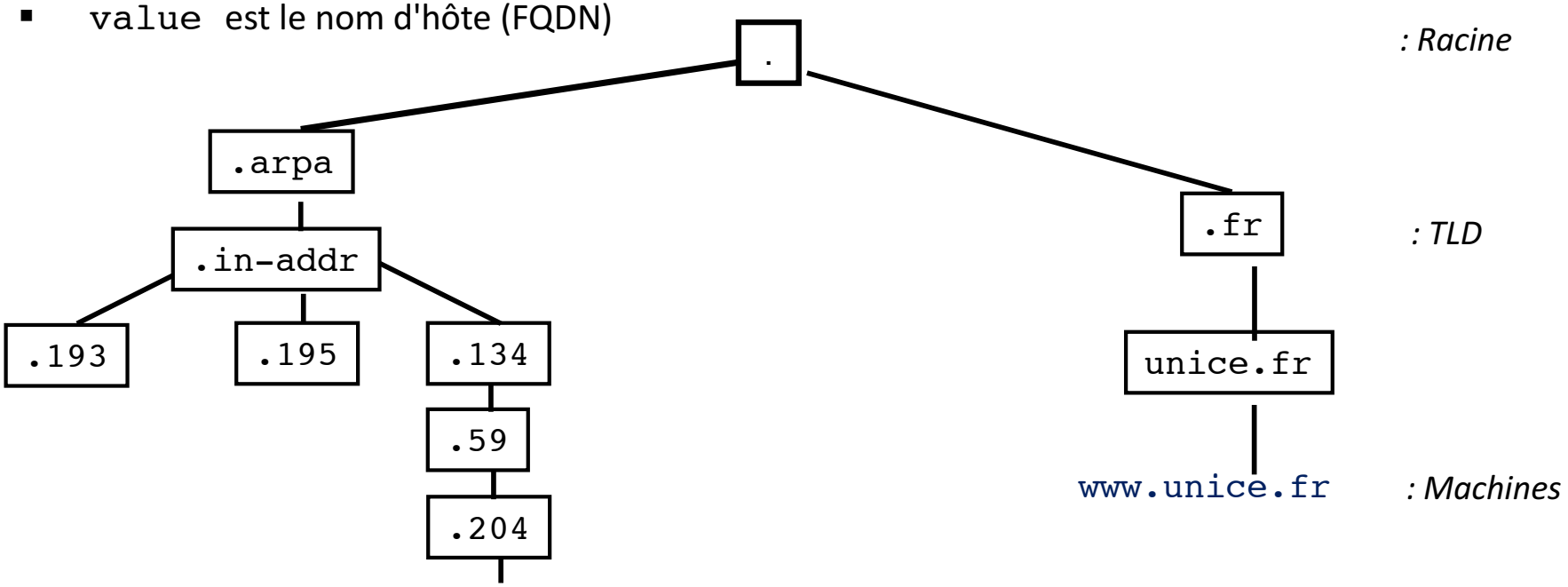
- Les attaques DDoS (*distributed denial of service attack*)
  - Bombarder serveurs racines avec du trafic
    - Filtrage du trafic
    - Serveurs DNS locaux mettent en cache adresses IP des serveurs TLD, permettant le contournement des serveurs racine
  - Bombard serveurs TLD
    - Potentiellement plus dangereux
- Rediriger attaques
  - *Man-in-middle* : intercepter des requêtes
  - Empoisonnement DNS : Envoyer fausse réponses sur le serveur DNS locaux, qui les mettent en cache
- Exploiter DNS pour DDoS
  - Envoyer des requêtes avec une adresse IP source usurpée: cible cette IP
  - Nécessite amplification

Reverse DNS

- **Résolution de nom inverse :** permet d'obtenir le FQDN (***fully qualified domain name***) d'un serveur à partir d'une adresse IP.
- ***Fully Qualified Domain Name*** (**FQDN**) : révèle la position absolue d'un nœud dans l'arborescence en indiquant tous les domaines de niveau supérieur jusqu'à la racine.

`www.unice.fr.`

- **Enregistrement *type = PTR***
  - `name` est l'adresse IP
  - `value` est le nom d'hôte (FQDN)



Vue d'ensemble: Web

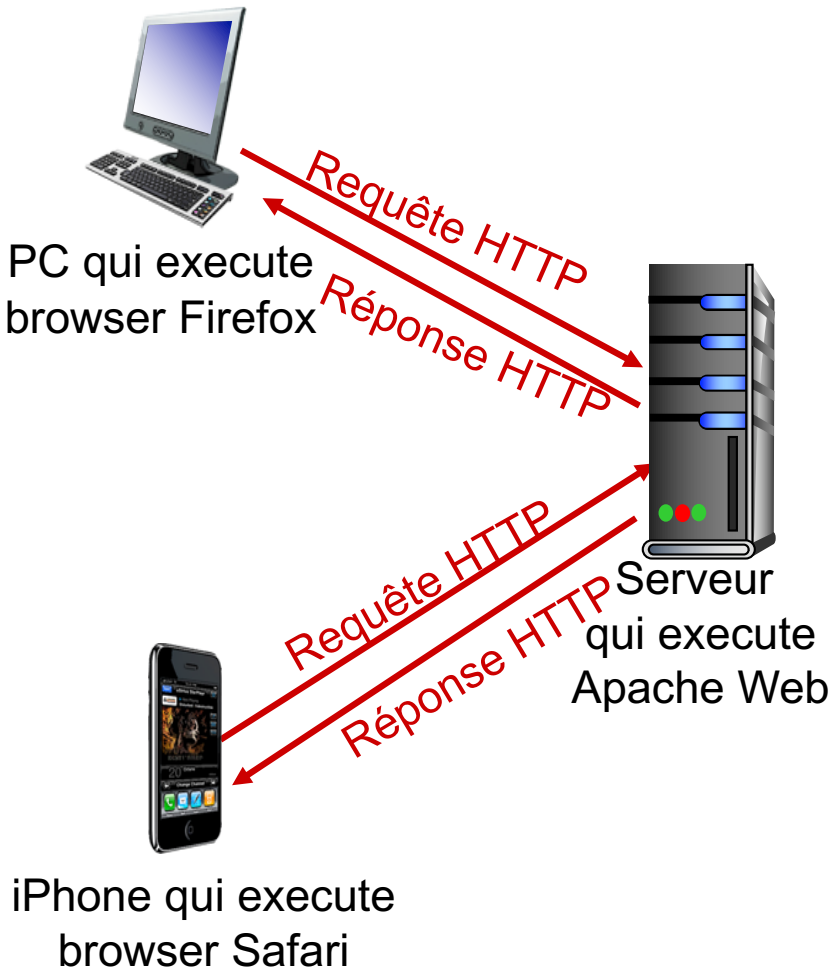
- Une page Web (si statique, elle est un document HTML) se compose d'objets.
- Un objet peut être un fichier HTML, une image JPEG, un applet Java, un fichier audio, ...
- Une page Web se compose d'un fichier de base HTML (`index.html`) qui comprend plusieurs objets référencés.
- Chaque objet est adressable par une adresse URL (*Uniform Resource Locator*), p. ex.:

Protocole : //	Hôte (nom domaine)	Chemin	Nom du ressource
`https://`	`www.unice.fr/`	`contenus-riches/images/`	`logo_UNS_haute_def.jpg`

Vue d'ensemble: Web

HTTP: *hypertext transfer protocol*

- Protocole d'application pour le service Web
- Modèle client-serveur:
  - **Client** : navigateur qui demande, reçoit, (en utilisant HTTP) et "affiche" objets Web
  - **Serveur** : serveur Web envoie (en utilisant HTTP) les objets en réponse aux demandes
- HTTP utilise TCP :
  1. Le client initie une connexion TCP (crée une socket) vers serveur web (port 80).
  2. Le serveur accepte la connexion TCP.
  3. Les messages HTTP sont échangés entre le navigateur (client HTTP) et le serveur Web (serveur HTTP)
  4. La connexion TCP est fermée

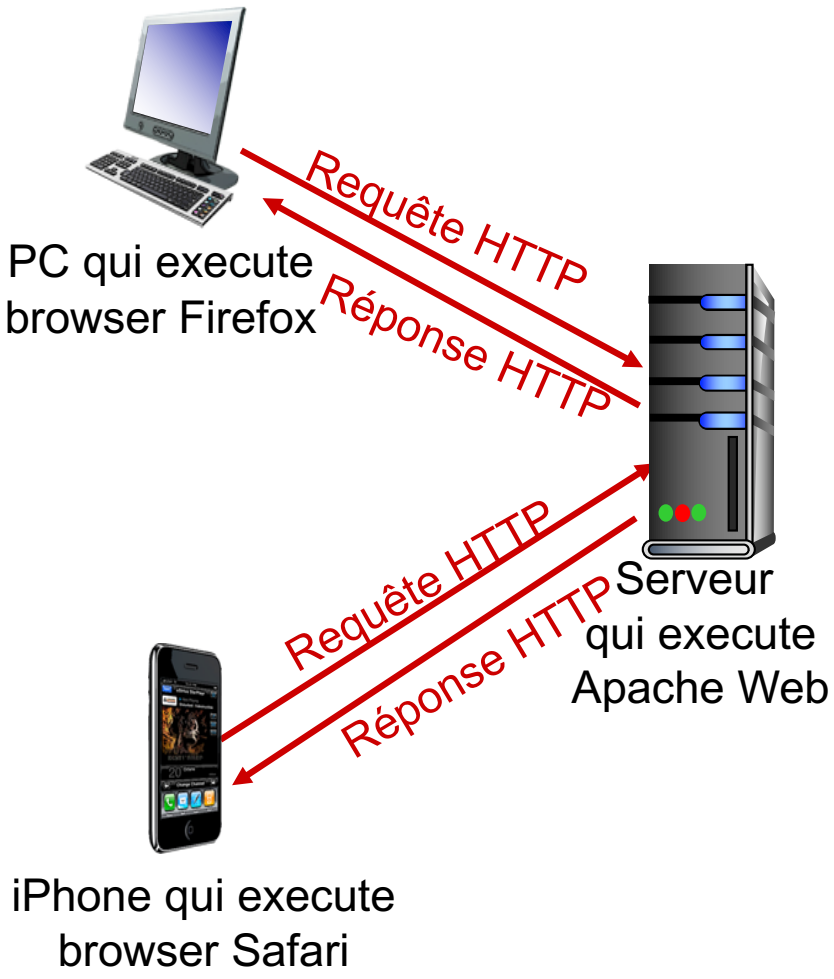


Vue d'ensemble: Web

HTTP: *hypertext transfer protocol*

- **HTTP est un protocole sans état: le serveur ne conserve aucune information sur les demandes passées des clients**

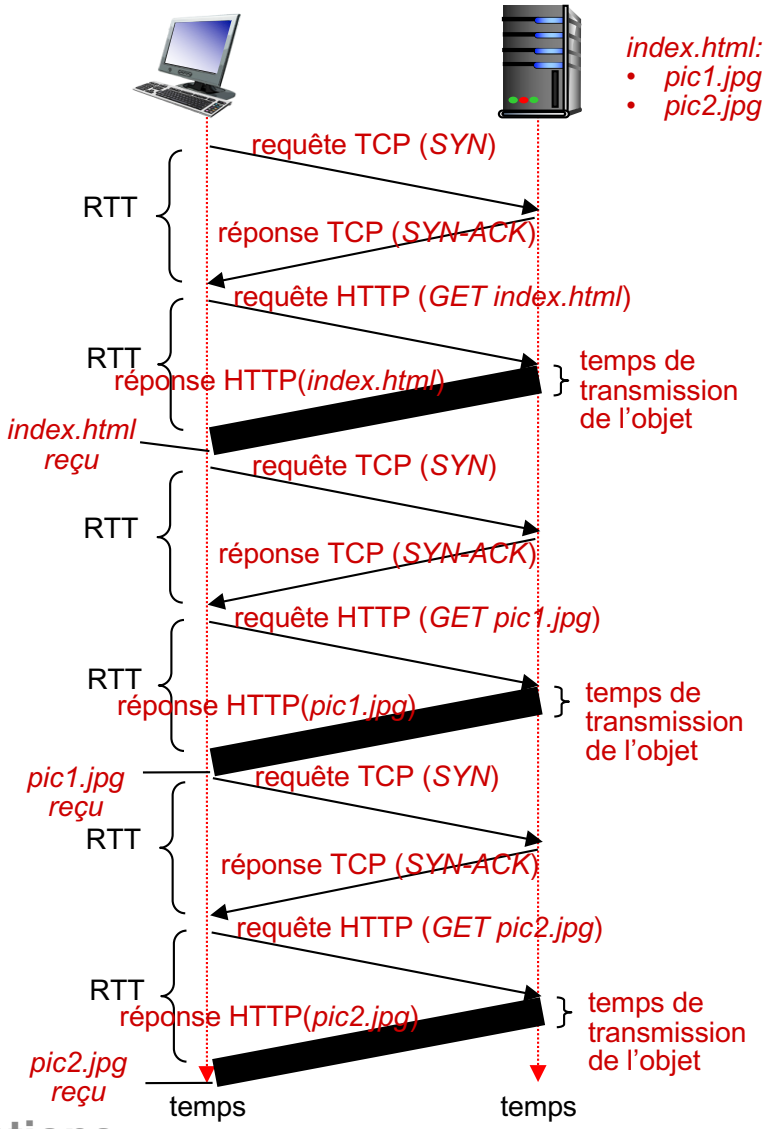
- **Protocoles avec état sont complexes! :**
  - Le passé doit être gardé.
  - Si le serveur / client se plante, leur points de vue de "l'état" peuvent être incompatibles.



HTTP non persistant vs persistant

- **HTTP non persistant** :
  - Au plus un objet envoyé via une connexion TCP ensuite fermée.
  - Le téléchargement de plusieurs objets requiert des connexions multiples.
  - Mode de connexion typique de **HTTP/1.0.**
  - Temps de réponse : (2RTT x nombre des objets) + temps de transmission des objets.
  - Surcharge du système d'exploitation pour chaque connexion TCP.
  - Les navigateurs ouvrent souvent des connexions TCP parallèles pour aller chercher les objets référencés.

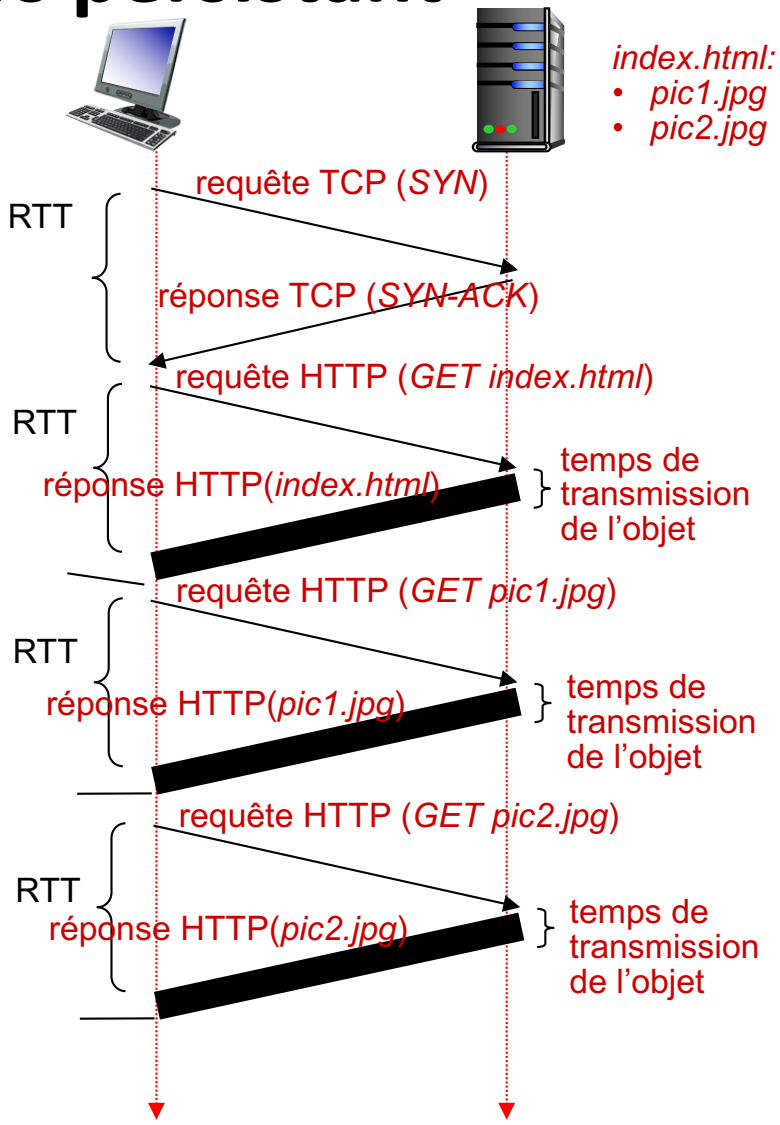
- **RTT (Round Trip Time)**
  - Le temps que met un paquet petit pour faire l'aller et le retour entre le client et le serveur



HTTP non persistant vs persistant

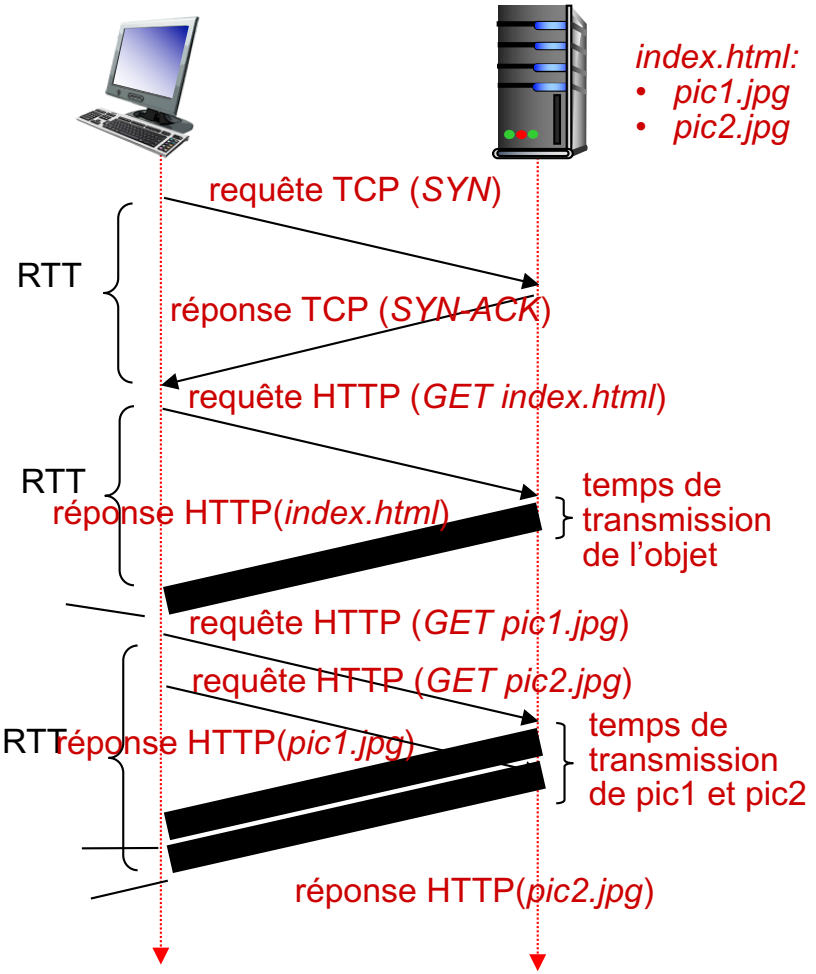
❖ **HTTP persistant** :

- Plusieurs objets peuvent être envoyées via une unique connexion TCP.
- Le serveur laisse la connexion ouverte après l'envoi de la première réponse HTTP.
- Les messages HTTP ultérieures sont envoyés par la connexion ouverte.
- Le client envoie des demandes dès qu'il rencontre un objet référencé.
- Mode de connexion typique de **HTTP/1.1** et **HTTP/2.0**
- Temps de réponse :
  - RTT x (nombre des objets+1) + temps de transmission des objets.



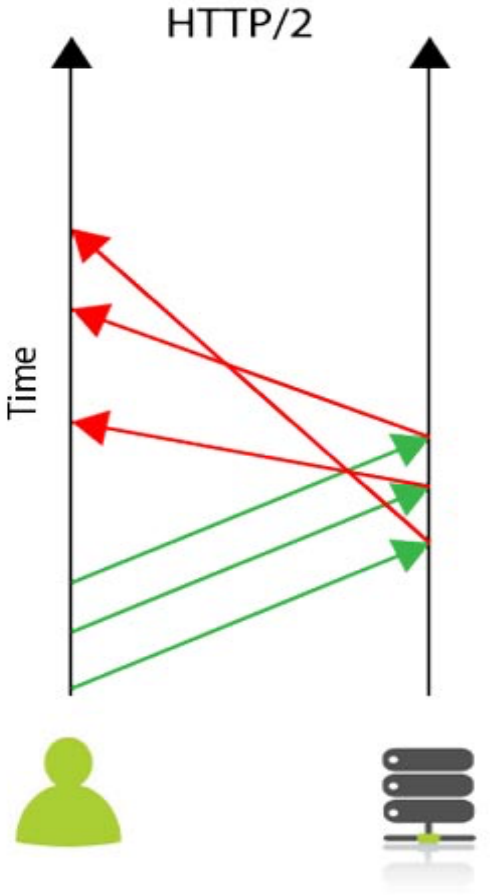
HTTP non persistant vs persistant

- **HTTP persistant** :
  - Avec *pipelining* : Plusieurs requêtes des objets référencés au même temps en parallèle :
    - Temps de réponse : approx. 3RTT + temps de transmission des objets.
  - *Pipelining* Synchrone
    - Mode typique de **HTTP/1.1**
    - Problème du blocage HOL (*head-of-line)*
  - *Pipelining* Asynchrone
    - Mode typique de **HTTP/2.0**
    - Le blocage HOL est éliminé, le site peut être chargé plus rapidement



HTTP non persistant vs persistant

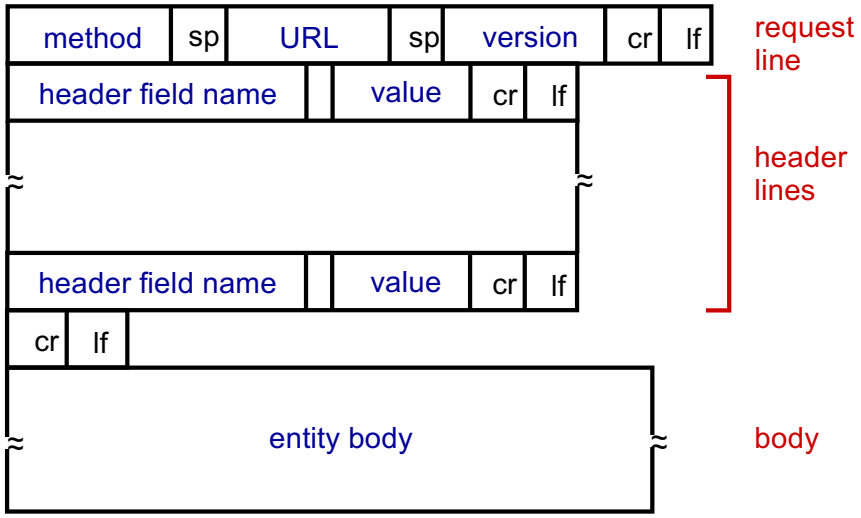
- **HTTP persistant** :
  - *Pipelining* Synchrone
    - Mode typique de **HTTP/1.1**
    - Problème du blocage HOL (*head-of-line)*
    - *Côté client,* avant qu'une demande ultérieure puisse être faite, les résultats de certaines demandes précédentes doivent avoir été reçus (les objets ont des priorités).
    - Le *serveur* doit envoyer ses réponses dans le même ordre que les demandes reçues: connexion reste FIFO (premier entré, premier servi)
  - *Pipelining* Asynchrone
    - Mode typique de **HTTP/2.0**
    - Le blocage HOL est éliminé, le site peut être chargé plus rapidement
    - *côté client,* le navigateur peut indiquer au serveur les priorités pour les objets. Les objets ayant des priorités similaires peuvent être groupés (multiplexés) dans la même requête.
    - *Au serveur,* le serveur peut envoyer les réponses dans un ordre différent que les demandes ont été reçues, mais en respectant les priorités.



Messages HTTP

Requête HTTP

```
GET /index.html HTTP/1.1 \r\n
Host: www-net.cs.umass.edu \r\n
User-Agent: Firefox/3.6.10 \r\n
Accept: text/html,application/xhtml+xml \r\n
Accept-Language: en-us,en;q=0.5 \r\n
Accept-Encoding: gzip,deflate \r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7 \r\n
Keep-Alive: 115 \r\n
Connection: keep-alive \r\n
\r\n
```



Messages HTTP

❖ Méthodes de la requête HTTP

Méthode	Description	Connexion HTTP
`GET`	Requête de la ressource située à l'URL spécifiée	`HTTP/1.0 et HTTP/1.1`
`POST`	Envoi de données au programme situé à l'URL spécifiée	`HTTP/1.0 et HTTP/1.1`
`HEAD`	Requête de l'en-tête de la ressource située à l'URL spécifiée	`HTTP/1.0 et HTTP/1.1`
`PUT`	Envoi de données à l'URL spécifiée	`HTTP/1.1`
`DELETE`	Suppression de la ressource située à l'URL spécifié	`HTTP/1.1`

❖ Envoyer de données au programme situé à l'URL :

- **Méthode POST** : Données envoyées vers le page web dans le corps de l'entité.
- **Procédé URL** : Données envoyées vers le page web dans le champ de la l'URL spécifié :

```
www.somesite.com/animalsearch?monkeys\&banana
```

Messages HTTP

Réponse HTTP

```
HTTP/1.1 200 OK \r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT \r\n
Server: Apache/2.0.52 (CentOS) \r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT \r\n
Accept-Ranges: bytes \r\n
Content-Length: 2652 \r\n
Keep-Alive: timeout=10, max=100 \r\n
Connection: Keep-Alive \r\n
Content-Type: text/html; charset=ISO-8859-1 \r\n
\r\n
<TITLE>Exemple</TITLE>
<P>Ceci est une page d'exemple.</P>
```

Messages HTTP

Réponse HTTP

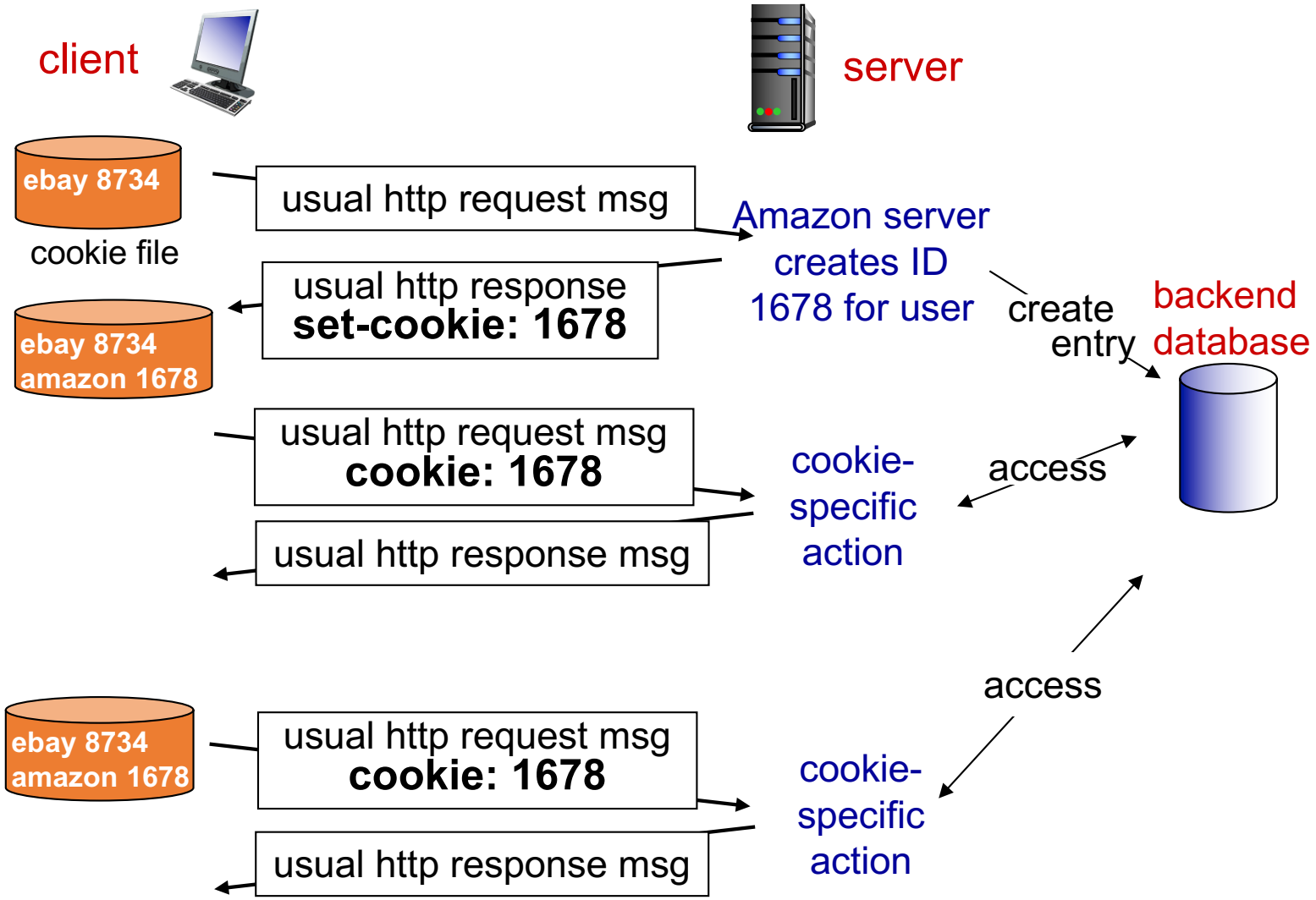
- Code d'état apparaît en 1ère ligne dans le message de réponse de serveur à client.

Code	Message	Description	Exemples
20x	Réussite	Ces codes indiquent le bon déroulement de la transaction	200 OK
30x	Redirection	Ces codes indiquent que la ressource n'est plus à l'emplacement indiqué	301 Moved Permanently
40x	Erreur due au client	Ces codes indiquent que la requête est incorrecte	400 Bad Request 403 Forbidden 404 Not Found
50x	Erreur due au serveur	Ces codes indiquent qu'il y a eu une erreur interne du serveur	505 HTTP Version Not Supported

Cookies

- De nombreux sites Web utilisent des cookies, mais un cookie, c'est quoi ?
  1. Un entête « cookie » dans le message de réponse HTTP
  2. Un entête « cookie » dans le dans le suivant message de requête HTTP
  3. Un fichier « cookie » conservé sur l'hôte de l'utilisateur, géré par le navigateur de l'utilisateur}
  4. Une base de données *back-end* sur le site Web avec tous les cookies
- Les cookies peuvent être utilisé pour garder:
  - **L'information personnel** des utilisateurs (p. ex. E-commerce) :
    - autorisation
    - paniers
    - recommandations
  - **L'état de la session** de l'utilisateur (p. ex. Web e-mail):
    - Les cookies permettent maintenir l'état à l'émetteur / récepteur sur plusieurs transactions
    - Les cookies portent l'état dans les messages HTTP

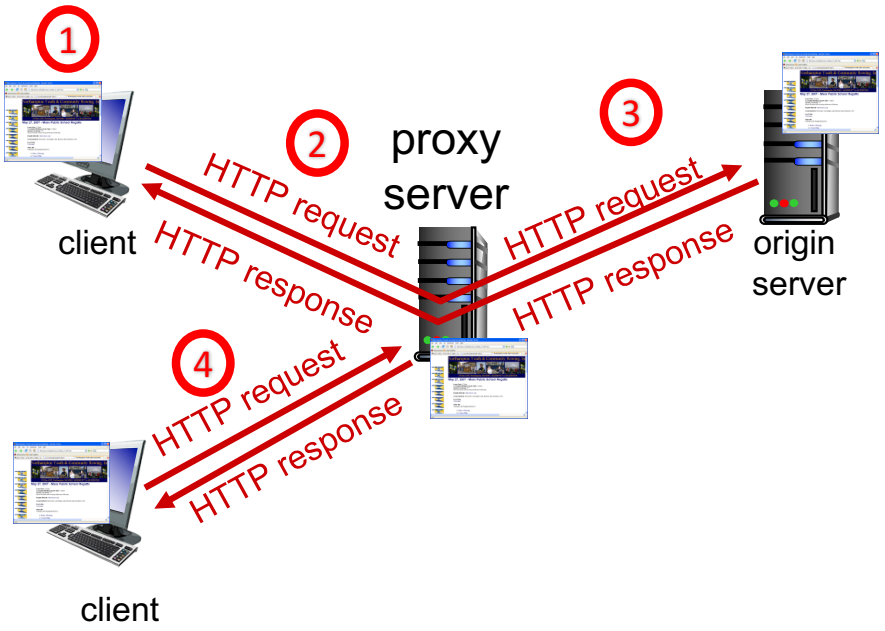
Cookies


client
server
ebay 8734
cookie file
usual http request msg
Amazon server
creates ID
1678 for user
create
entry
backend
database
usual http response
set-cookie: 1678
ebay 8734
amazon 1678
usual http request msg
cookie: 1678
cookie-
specific
action
access
usual http response msg
access
ebay 8734
amazon 1678
usual http request msg
cookie: 1678
cookie-
specific
action
usual http response msg

Caches Web

- **Idée** : Satisfaire la requête du client original sans impliquer le serveur Web d'origine, en utilisant un *cache local* ou un *serveur proxy* qui sert à la fois comme client et comme serveur (1) serveur pour le client original et (2) client pour le serveur d'origine

1. Le navigateur est configuré pour accéder à la Web via cache (local ou serveur proxy)
2. Le navigateur envoie toutes les requêtes HTTP à cache
3. S'il objet n'est pas dans la cache, on va le chercher au serveur Web d'origine, et on le garde dans la cache
4. Après, s'il objet est déjà dans la cache, il est rendu directement.



- Pourquoi la mise en cache Web?
  - réduire le temps de réponse pour la demande du client
  - réduire le trafic sur la liaison d'accès de l'institution
  - Internet dense avec caches: permet aux fournisseurs de contenu «pauvres» pour fournir efficacement leur contenu (aussi P2P)

Caches Web (GET conditionnelle)

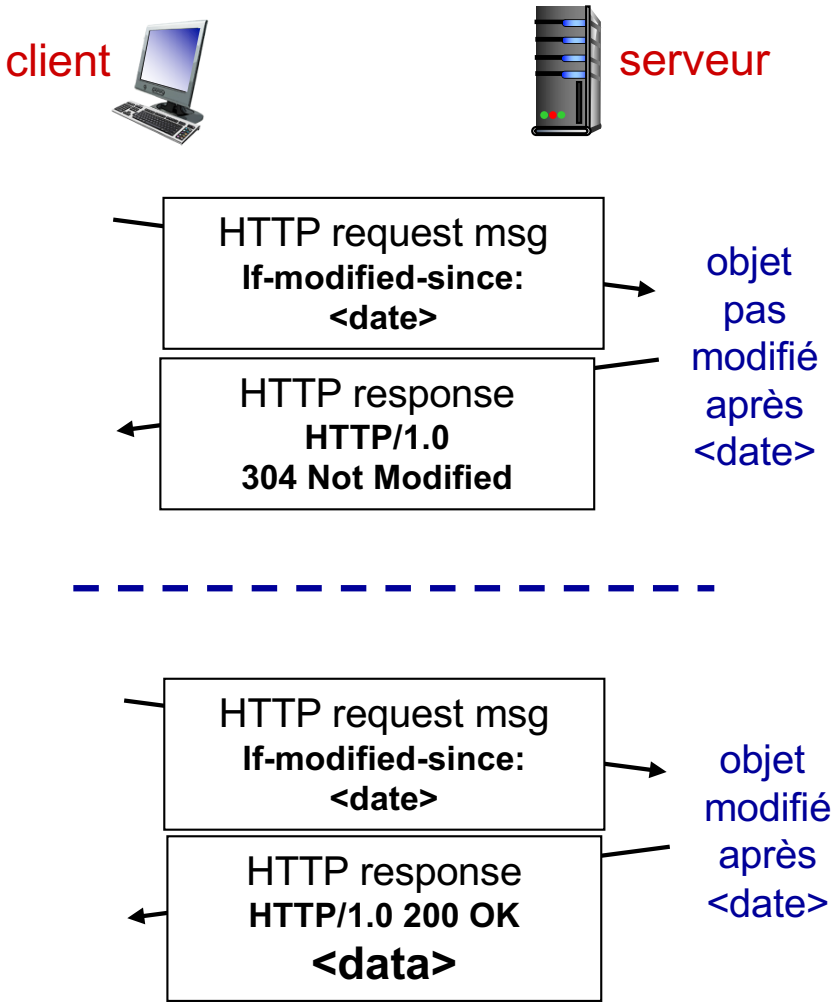
- **Objectif**: ne pas envoyer un objet si le cache a déjà une version mise à jour de l'objet :
  - pas de délai de transmission de l'objet
  - utilisation inférieure du liens

- *Cache* : Elle écrit la date de sa copie mise en cache dans l'entête de la requête HTTP

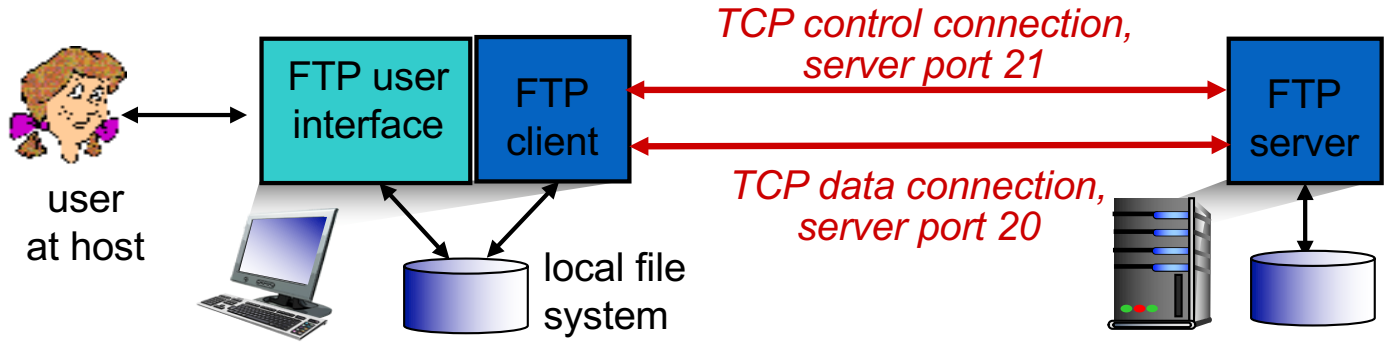
```
If-Modified-Since: <date>
```

- *Serveur* : Réponse HTTP ne contient aucun objet si la copie en cache est encore à jour

```
HTTP / 1.0 304 Not Modified
```



FTP : Vue d'ensemble

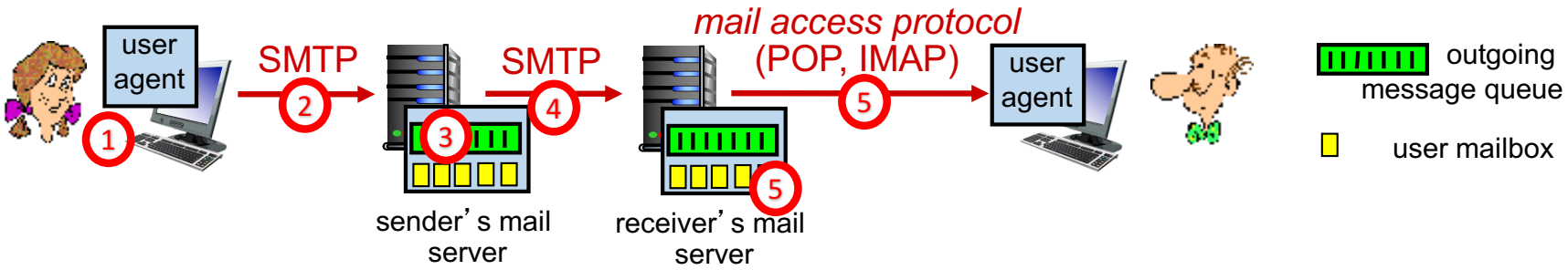


- FTP (RFC 959): transfert de fichier de/vers un hôte distant, modèle client-serveur
- Deux connections TCP :
  - Connexion de contrôle (« hors bande ») sur le port 21
  - Connexion de données sur le port 20
- Echanges des messages :
  1. Le client FTP contacte le serveur FTP sur le port 21, en utilisant TCP
  2. Le client est autorisé sur la connexion de contrôle
  3. Le client navigue dans le répertoire distant (commandes sur le canal de contrôle)
  4. Lorsque le serveur reçoit la commande de transfert de fichiers, le serveur ouvre une 2è connexion de données TCP (pour le fichier) vers le client
  5. Après le transfert d'un fichier, le serveur ferme la connexion de données
  6. Le serveur ouvre une autre connexion de données pour transférer un autre fichier
- Le serveur FTP maintient « l'état » : répertoire courant, l'authentification préalable

Commandes et réponses FTP

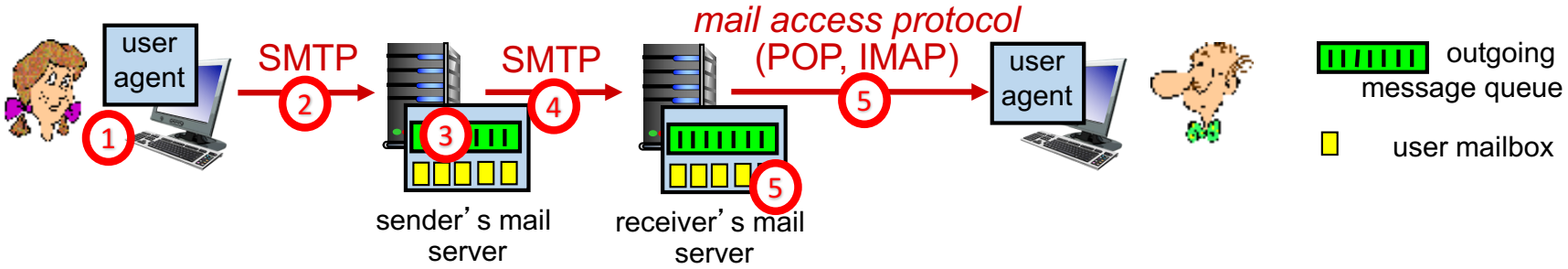
- **Commandes** : envoyés en tant que texte ASCII sur le canal de contrôle
  - `USER username`
  - `PASS password`
  - `LIST` : récupère la liste des fichiers dans le répertoire
  - `RETR` filename : récupère le fichier
  - `STOR` filename : stockes le fichier sur l'hôte distant
- **Codes de retour** : code d'état (comme dans HTTP)
  - `331 Username OK, password required`
  - `125 data connection already open; transfer starting`
  - `425 Can't open data connection`
  - `452 Error writing file`

Courrier électronique: Vue d'ensemble



- Les agents utilisateurs (*user agents, UA*)
  - Composition, édition et lecture des messages électroniques
  - Les messages entrants, sortants stockés sur le serveur (OU PAS!!!)
  - P. ex., Outlook, Thunderbird, le client de messagerie iPhone
- Les serveurs de messagerie
  - Boîte aux lettres qui contient des messages entrants pour l'utilisateur (*user mailbox*)
  - File de sortie des messages électroniques à envoyer (*outgoing message queue*)
- Les protocoles:
  - Envoyer: échange et stockage des e-mails entre les serveurs de messagerie:
    - SMTP: Simple Mail Transfer Protocol [RFC 2821]
  - Recevoir: accès aux serveurs de messagerie pour récupérer les e-mails
    - POP: Post Office Protocol [RFC 1939]
    - IMAP: Internet Mail Access Protocol [RFC 1730]
    - HTTP: Gmail, Hotmail, Yahoo! Mail, etc.

Courrier électronique: Vue d'ensemble



1. Alice utilise l'UA pour composer un message pour `bob@unice.fr`
2. L'UA d'Alice envoie un message à son serveur de messagerie, le message est mis en la file de sortie de messages
3. Le serveur SMTP de messagerie d'Alice ouvre une connexion TCP (comme client) vers le serveur SMTP de messagerie de Bob
4. Le serveur SMTP d'Alice envoie le message d'Alice sur la connexion TCP
5. Le serveur SMTP de messagerie de Bob place le message dans sa boîte aux lettres
6. Bob invoque son agent utilisateur pour récupérer et lire le message, en utilisant POP, IMAP ou HTTP

SMTP

Session SMTP

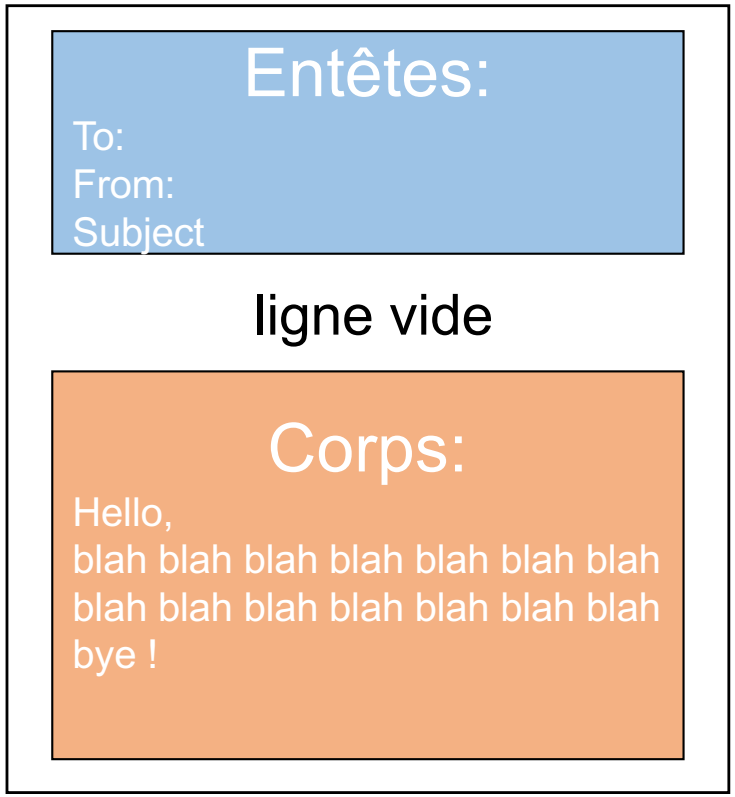
- SMTP utilise TCP pour transférer les courriels de manière fiable (port 25)
- Une session est une succession des interactions commande/réponse (HTTP, FTP ...) :
  - commandes (texte ASCII) envoyées par le client (HELLO, MAIL FROM, ...)
  - réponses (code d'état et phrase) envoyées par le serveur pour indiquer le résultat des commandes
- Trois phases de la session: (1) poignée de main, (2) transfert, et (3) fermeture
- SMTP utilise des connexions persistantes (fermeture à la fin de la session)
- Serveur SMTP utilise `CRLF.CRLF` pour déterminer la fin du message!

Client	Serveur
	220 smtp.unice.fr
HELLO smtp.fai.fr	250 Hello smtp.fai.fr, pleased to meet you
MAIL FROM : <expediteur@fai.fr>	250 sender <expediteur@fai.fr>... ok
RCPT TO : <destinataire@unice.fr>	250 recipient <destinataire@unice.fr>... ok
DATA	354 go ahead
Hello, blah blah blah blah blah blah blah blah blah blah blah blah blah blah bye ! .	250 ok : Message 94953219 accepted
QUIT	221 smtp.unice.fr closing connection

SMTP

Format des messages

- RFC 822: norme de format de messages
- Les messages (en-tête et le corps) doivent être en 7-bit ASCII.
- Lignes d'en-tête, par exemple,
  - To :
  - From :
  - Subject :
- Corps (« message ») : uniquement des caractères ASCII
- ATTENTION : Lignes d'entête sont différents des commandes SMTP FROM, RCPT TO !



Protocoles d'accès à la messagerie

- **POP3** :
  - POP3 est sans état
  - Deux phases : autorisation et transfert
  - Deux modes :
    - *download-and-keep* : une copie des messages sur différents clients
    - *download-and-delete* : on ne peut pas relire l'e-mail en cas de changement client, ex:

Client	Serveur
	+OK POP3 server ready
user moi	+OK
pass 1234	+OK user successfully logged on
list	1 67989
	2 90091
	.
retr 1	<message 1 contents >
.	
del 1	
retr 2	<message 2 contents >
.	
del 2	
quit	+OK POP3 server signing off

- **IMAP** :
  - IMAP conserve tous les messages en un seul endroit: au serveur
  - IMAP permet à l'utilisateur d'organiser les messages dans des répertoires
  - IMAP maintient l'état utilisateur d'une session: (1) les noms de dossiers, et (2) les correspondances entre les ID de messages et le nom des dossier